Panel: Ada and High Performance Computing—Reality or Myth?

Chair
Steven Deller, Verdix Corporation

Panelists
Philippe Collard, Telesoft Corporation
Anthony Gargaro, Computer Sciences Corporation
Dr. Stephen Zeigler, Verdix Corporation
Dr. Thomas H. Probert, Center for High Performance Computing

As we enter the 1990's, new technologies for high performance computing are rapidly becoming available. Single RISC processors are already available with performance exceeding 100MIPS and 40MFLOPS. By the end of the decade processing performance exceeding 10GIPS and 1GFLOPS seem likely.

More importantly, multiprocessor chips and multiprocessor operating systems are starting to become widely available, promising inexpensive multiprocessor support for day to day applications.

These Multiple Instruction Multiple Data (MIMD) systems seem destined to be the mainstay of high performance computing by the end of the decade if not sooner. There are certainly other forms of high performance computing available today, but the performance increases coming from these MIMD systems, the wider applicability of these systems, and ongoing announcements of multiprocessing workstations from major vendors suggest that MIMD systems will be the major high performance computing platforms.

The natural question to ask is whether Ada is capable of taking advantage of the new performance capabilities made available by MIMD systems. This is not a simple question to answer. Any language is, to some degree, capable of using multiprocessing. The heart of the question is whether a programmer, using Ada, can naturally take advantage of a multiprocessor system and still keep the program integrity and software engineering paradigms for which Ada was designed.

A number of specific questions surface, such as:

- Do the abstraction mechanisms used in software engineering hinder the use of high performance systems?
- Does Ada tasking fit well with MIMD systems?
- Will the new Ada9X synchronization mechanisms be compatible with MIMD operating systems?
- Does Ada subprogram nesting and globally visible data make using MIMD systems harder?

The panelists have been selected for their insights into these and other issues relating high performance computing to Ada. Each has worked for a number of years with high performance computing solutions in Ada and should bring

To prepare for this panel, each panelist was asked to provide a short position paper that addressed the following questions:

1. Is high performance computing fundamentally compatible or incompatible with Ada?
2. What work needs to be done to make high performance computing with Ada a reality?
3. Will Ada9X have any significant impact on high performance computing and if so, how?
4. When will high performance computing with Ada be a reality?

The panelists responses to these questions and to any other issues relating to Ada and high performance computing follow.

Position Paper: Ada and HP Computing

Philippe Collard
Telesoft Corporation

Panel members were asked to answer four questions. Before addressing these questions, it seems necessary to comment on the term "High Performance Computing".

In the late 1970s, a company called Cray Research validated the notion of "supercomputer" with the introduction of the Cray 1. The machine seemed awesomely powerful at the time. And was awesomely expensive.

Less than 20 years later, Digital Equipment's Alpha microprocessor embeds, on a single chip, a computing power equivalent to that of the Cray 1. The expected cost of an Alpha workstation is less than $20,000. It is thus rather obvious that the notion of "high performance" is a very fast moving target. Hence, I have deliberately elected to limit the scope of this paper to the very high end of the high performance domain: supercomputers in all shapes, sizes, architect

1. Is high performance computing fundamentally compatible or incompatible with Ada?

FORTRAN is the most widely used language in the HPC community. In the age of object-oriented programming, CASE tools, and distributed computing, a language developed three decades ago is still the most widely used vehicle for programming high-performance computers. FORTRAN's mutations to keep up with the latest in supercomputing are simply amazing. The latest incarnation, FORTRAN 90, attempts to address parallel processing from MIMD to SIMD architectures: not a small feat!

From this we can deduce that assessing compatibility or applicability of a particular programming language to a particular domain of computing is fairly subjective. Therefore Ada is not more or less compatible or incompatible with HPC than say C, C++ or even BASIC. However, Ada possesses a few characteristics that set it apart from other languages.

First Ada is a very tightly controlled standard. Second Ada offers the possibility of expressing parallelism at the language level (via tasking). Third Ada recognizes the need for language features addressing numerical computations (though this is probably the most intricate part of the language to understand).

With these three attributes, Ada has as good a chance as any other language to replace FORTRAN or at least to "compete" with FORTRAN in the HPC domain.

2. What work needs to be done to make high performance computing with Ada a reality?

The use of Ada in the HPC community is limited by two factors. One is the availability of robust, well tuned compilers on traditional and less traditional high performance computing platforms (within the scope of this paper). Then there is the large body of existing FORTRAN code that constitutes the bulk of the HPC application software.

To make Ada a contender in the HPC community, we, the Ada community must do three things. First we must increase the availability of good Ada compilers on HPC platforms. The diversity of HPC architectures will certainly be a challenge. We should focus on the architectures that are relatively well matched with the Ada model (i.e. MIMD machines).

Second we must educate the next generation of HPC users. These future users are still in universities and colleges. They learn to program in FORTRAN and C. Let us provide them with the tools to learn in Ada.

Third, we must demonstrate that Ada may provide what is most important to HPC users: performance. The work described in [1] provided such a demonstration. More examples are needed to make Ada a language of choice in the HPC community.

3. Will Ada9X have any significant impact on high performance computing and if so, how?

Ada was a very ambitious project since it aimed not only at defining a programming language but also a model of computing. In our era of "open everything", one has to wonder if rigid, all encompassing models are still necessary. Yet, the two most important characteristics for a standard are implementability and usability.

Thus the question is not to determine if Ada 9X is the right vehicle to impact HPC but rather if Ada 9X can encapsulate efficient tools that

1. will be implemented by vendors at a reasonable cost and
2. will be efficiently utilized by HPC users.

TeleSoft has developed an advanced prototype of a distributed Ada development system targeted at loosely coupled multiprocessor architectures. This system (that will be demonstrated at Tri- Ada'92 [2]) could be used to program computers with a few to a few hundreds processors. It is largely based on a draft (August 1991) of the Ada 9X recommendation for distribution of Ada programs.

One of the most salient features of this system is the graphical user interface which allows users to same time, there are horror stories of huge, incomprehensible FORTRAN programs buried in the heart of critical systems. At some point these programs will be re-written. If common sense rules, care will be taken to re-engineer these programs using modern tools and techniques. Ada should be a strong contender.

3. Will Ada9X have any significant impact on high performance computing and if so, how?

Several features of Ada9X will improve Ada's viability in HPC. Most important among these are the object-oriented extensions. These extensions are NOT a panacea, but are an important tool for prototyping and "organizing" programs; without these extensions to Ada, C++ might have the FORTRAN replacement field locked up by default. The proposed Ada9X changes for distributed programming are probably premature since there is little experience in this area. The proposals for parallelizing loops seems to leverage the experience with FORTRAN in loop parallelization. The proposed "refurbishment" of Ada float I/O was a bit esoteric and is hard to assess; in any case the FORTRAN floating point seems to get by with little semantic rigor.

While Ada9x may not be crucial to FORTRAN replacement, it will help.

4. When will high performance computing with Ada be a reality?

Ada will take an increasingly important role in HPC because of its tasking and because many hardware vendors are producing multiprocessors. The acceptance of Ada in the traditional FORTRAN areas of HPC will depend on accomplishing student/teacher support, HPC library support, host/target support and comparable performance.

Compiler vendors should be able to satisfy these requirements by mid-1994, other than HPC library support. Government encouragement would help in creating Ada rewrites and interfaces for common HPC packages.

Position Paper: Ada and HP Computing

Dr. Thomas Probert
Center for High Performance Computing

Ada and High Performance Computing: A Window of Opportunity?

High performance computing, while not limited to supercomputing, inherits its legacy. This includes the problem solving paradigms, hardware architectural and software tools as well as a mind set. Ada, like any other language, can be used to develop high performance applications given the existence of high performance compilation and run time systems.

Ada, in its early years focussed on standards conformance to ensure portability. This achieved compatibility at the expense of performance. That decision has implications for Ada's role in high performance computing today.

In addition, the notion of developing tools and environments to support, and in some cases enforce, software engineering principles was done more or less in a vacuum. Whether Ada has been a successful experiment in attaining the lofty goals of improving the state of software engineering practice is still to be decided but one result is clear; the Ada community, like the Common Lisp community, stands isolated from the mainstream of information processing. This is true of the high performance computing community as well.

Large amounts of Ada parallel processing software, developed on sequential or networked sequential architectures under the idea of "software first" and "software reuse" will be virtually unusable on high performance machines. Will these large applications be re-written or will those applications not use parallel architectures?

The question is not whether Ada can be useful in high performance community but will it be used?

This talk will focus on this basic question and the other central issues that need to be addressed in order to capture at least part of the high performance computing market.

Position Paper: Ada and HP Computing

Dr. Stephen Zeigler
Verdix Corporation

1. Is high performance computing fundamentally compatible or incompatible with Ada?

High Performance Computing (HPC) is not incompatible with Ada. That is, there is nothing fundamental in the design of Ada that precludes the highest levels of optimization. Indeed, Ada has strengths in many areas, including tasking, floating point, machine access, and modular programming definitions, that make it technically the best language for HPC.

Compared to FORTRAN, Ada's definition is consistent and portable, with strong support for modern programming styles. Ada's support from government, as for example via the ACVC testing suites, has resulted in relatively portable, predictable programming for its users.

However, Ada has practical problems that must be overcome before it can displace FORTRAN as the HPC language of choice.

2. What work needs to be done to make high performance computing with Ada a reality?

Ada will displace FORTRAN only when most of the following conditions are met:

- Ada is taught to engineering students, which requires that Ada have student toolsets at student prices, and that teachers have teaching tools for Ada in HPC.
- Most HPC computing libraries are accessible via Ada interfaces or are rewritten in Ada.
- Good Ada tools are available for popular host/target hardware and operating systems.
- Ada's perceived performance is roughly similar to that delivered by FORTRAN.

None of these conditions are met today.

Ada tools are too expensive and use too much disk space for most student budgets, and there are no good curriculum materials to support HPC in Ada. There are many FORTRAN packages that have no Ada interfaces. There are major HPC platforms such as the Touchstone, the Thinking Machines, the MAICO, the NCube, and even the Cray where there is no Ada or the Ada is viewed as unreliable or unfriendly. Today, Ada compilers are not perceived as delivering the pure performance of Fortran.

This last point, the performance of generated Ada code, deserves some comment. FORTRAN has been around for 35 years. It has many optimizations. It also has a user community that is trained to work around its many idiosyncracies to work with the compilers for maximum performance. It has a suite of benchmarks that are tuned to enable maximum Fortran performance.

In apples-to-apples performance Ada is probably not dissimilar from FORTRAN and may even exceed its performance where Ada types and tasking information can be used for optimization. However, Ada does need further work:

- **Benchmarks must be written to enable maximum Ada performance.** This means avoiding "FORTRAN-isms" such as unchecked conversions to mimic common blocks, or casting every data type as an array. It also means avoiding unnecessary use of expensive Ada constructs such as arrays with dynamic lower bounds.
- **Ada compilers must mature in their support of HPC coding techniques,** especially including vectorization, software pipelining, strip mining and loop inversion/alteration. In addition, Ada compilers must improve their support of exception-less optimizations.
- **Ada code generators must supply more machine-dependent optimizations for the HPC architectures.** Here, time is an advantage for Ada. The new architectures represent substantially improved performances for small chips, bring their capability into the supercomputer levels even as single chips. The new architectures often require as much work for FORTRAN compiler writers as for Ada, so increasingly FORTRAN's optimization leads will wither. In addition, the tasking support for Ada will make it much more attractive on multiprocessors, at least outside highly specialized parallel algorithms.

Ada also has strengths beyond its code generation and tasking—Ada is the basis for a software development tool set of which the compiler is only a part. Until now, the software engineering strengths of Ada have been ignored or minimized by engineers in HPC. At the decompose applications in an interactive ("point and click") manner. Yet it is built out of "stock" components. In other words, no special version of the compilers, linkers or any other standard tools was necessary to assemble this system.

We consider this effort as a demonstration that it is indeed possible to include paradigms such as distributed programming in the Ada standard and still leave the language definition open enough that a variety of implementations are possible for a variety of architectures. For all these reasons we encourage the work that is now going on in Ada 9X for features related to high-performance computing.

4. When will high performance computing with Ada be a reality?

High-performance computing with Ada will become a reality when a significant number of users recognize that the Ada language has many features well suited to HPC-type applications than cannot be found in other languages. This could occur sooner than one may think.

I recently attended a users group meeting of a fairly large computer manufacturer headquartered in the San Francisco bay area. This company is involved in the release of a new version of their base operating system where the notion of multiprocessing is very important. The basic mechanism used to control multiprocessing at the user level is not too distant from the tasking model.

During this meeting, I heard engineers from this company (a stronghold of the C culture) making statements about how Ada is so interesting from the multiprocessing point of view! Ada will become a reality in the HPC area when people start "hacking" with Ada tasking for programming parallel applications.

References:

[1]P. Collard, A. Goforth, M. Marquardt. Ada as a Parallel Language for High Performance Computers: Experience and Results. TRI-Ada'90. December 1990. Baltimore.

[2] T. Burger, R. Volz & al. A Practical Tool for Distributing Ada Programs: TeleSoft's Distributed Ada Configuration Tool. To be presented at TRI-Ada'92. November 1992. Orlando.

Position Paper: Ada and HP Computing

Anthony Gargaro
Computer Sciences Corporation

The notion of high performance computing typically includes a number of distinct technologies such as parallel processing and distributed execution. These technologies require a level of specification of the target architecture that may lead to linguistic compromises in a common programming language.

In the last decade there is evidence that Ada has been both successful and unsuccessful in developing applications where these technologies have been important. Therefore, one's perspective of the "reality or myth" of Ada to develop high performance computing applications may depend upon the specific criteria demanded by the application.

For example, Ada is used today in distributed execution environments. In some instances, the purpose for this distribution is to improve performance, where high performance not only requires reducing processing time, but increasing reliability or fault tolerancy. A system that can continue execution in the presence of degraded conditions, without suffering costly restart, exhibits a kind of performance vital to some application domains. It may be contended that in these systems, high performance has been achieved by advancing beyond what is specified in the RM, e.g., multiple main subprograms, or that a fundamental tenet of the language, e.g., type safety, has been jeopardized. In other words, the language has been compromised either by permissions or restrictions supported by an implementation. Such a contention often gives credence to the argument that a programming language should include features for high performance computation regardless of the target architecture; it is this argument that promotes debate. The "reality" is that a common programming language should offer the opportunities for innovative implementation without undermining the rationale for the language.

The proposed Ada 9X mapping refines this "reality". The rapid change in hardware and software occurring since the standard is recognized as continuing during the life of the revised standard. Consequently, the mapping emphasizes the Ada tradition of exploiting straightforward semantic models for parallel processing and distributed execution by including implementation guidance and options for specialized application domains. This emphasis, together with enhanced features, liberates Ada so that it may become the preferred language for integrating high performance software components.